

DATA STRUCTURE AND ALGORITHM

LAB -1



August 24, 2025

Sharjeel Memon (24k-0555)

Lab Tasks

# Q1. Bank Account Management System

Suppose you are developing a bank account management system, and you have defined the BankAccount class with the required constructors. You need to demonstrate the use of these constructors in various scenarios.

1. Default Constructor Usage: Create a default-initialized BankAccount object named account1. Print out the balance of account1.

2. Parameterized Constructor Usage: Create a BankAccount object named account2 with an initial balance of $1000. Print out the balance of account2.

3. Copy Constructor Usage: Using the account2 you created earlier, create a new BankAccount object named account3 using the copy constructor. Deduct $200 from account3 and print out its balance. Also, print out the balance of account2 to ensure it hasn't been affected by the transaction involving account3.

# Q2. Exam Class with Dynamic Memory Allocation (DMA)

Create a C++ class named "Exam" using DMA designed to manage student exam records, complete with a shallow copy implementation. Define attributes such as student name, exam date, and score within the class, and include methods to set these attributes and display exam details.

As part of this exercise, intentionally omit the implementation of the copy constructor and copy assignment operator. Afterward, create an instance of the "Exam" class, generate a shallow copy, and observe any resulting issues.

# Q3. Box Class with Dynamic Memory Allocation and Rule of Three

Create a C++ class Box that uses dynamic memory allocation for an integer. Implement the Rule of Three by defining a destructor, copy constructor, and copy assignment operator. Demonstrate the behavior of both shallow and deep copy using test cases.

Solutions

# Q1. Bank Account Management System

#include <iostream> // Easy Access to Input and Output Operations

using namespace std; // So we dont have to write "std::" before every library features (like cout,cin,endl ,etc)

class BankAccount { // Defining the BankAccount Class

private : // Access Modifier , private, makes member accessible only inside the class

double Balance; // Data Member

public : // Access Modifier , public, allows functions to be accessible from outside

//Constructors

BankAccount () : Balance(00.00) { // Default Constructor - Initializes Account with $0 if no paramater is passed!

cout<<"Account Created!"<<endl;

}

BankAccount (double x) : Balance(x) { // Parameterized Constructor - Initializes Account's Balance with the paramater passed!

cout<<"Account Created!"<<endl;

}

BankAccount(const BankAccount &other) { // Copy Constructor - Used to Create a copy of another Object

cout<<"Account Created!"<<endl;

Balance = other.Balance;

}

// Methods

void withdraw(double x) { // Withdraw method - Deducts the parameter from the Balance

if( Balance >= x ) { // Check if Balance is either greater than or equal to parameter

Balance -= x; // Deducting paramter from the Balance

cout<< "Amount Deducted Successfully!"<<endl;

return ; // End of Function

}

cout<< "Insufficient Balance!"<<endl; // If not Sufficient Balance

return ;

}

void deposit(double x) { // Deposit Method - Adds Paramter to the Balance

if(x > 00.00) { // Checks if the Parameter is Positive

Balance+=x;

cout<< "Amount Deposited Successfully!"<<endl;

return ; // End of Function

}

cout<< "Deposit can't be less than or equal to 0!"<<endl; // Invalid Parameter

}

double getBalance() const { // Getter Method - Allows main to Access the Private Members

return Balance;

}

};

int main() {

// Default Constructor Usage

BankAccount account1; // Will Initialize with $0 Balance

cout<<"Account1 Balance : $"<<account1.getBalance()<<endl<<endl; // Prints Balance for Account1

// Parameterized Constructor Usage

BankAccount account2(1000.00); // Will Initialize with $1000 Balance

cout<<"Account2 Balance : $"<<account2.getBalance()<<endl<<endl; // Prints Balance for Account2

// Copy Constructor Usage

BankAccount account3 = account2; // Will Initialize Balance of Account3 with the Balance of Account2

account3.withdraw(200.00); // Deducting $200.00 from Account3

cout<<"Account3 Balance : $"<<account3.getBalance()<<endl<<endl; // Prints Balance for Account3

cout<<"Verifying to Show that the Balance of Account2 is Unchanged..."<<endl;

cout<<"Account2 Balance : $"<<account2.getBalance()<<endl<<endl; // Prints Balance for Account2

return 0; //End of Program

}

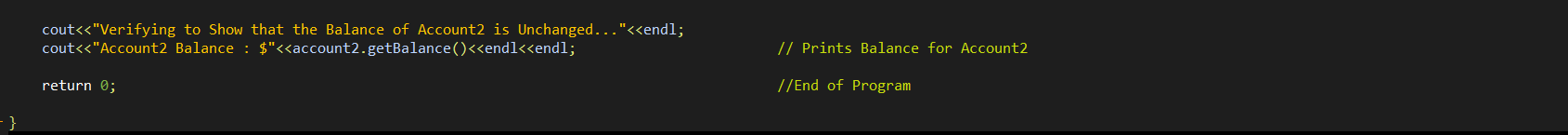
# Q1. Screenshots

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer screen

AI-generated content may be incorrect. A screenshot of a computer program

AI-generated content may be incorrect. 

# Q2. Exam Class with Dynamic Memory Allocation (DMA)

#include <iostream> // Easy Access to Input and Output Operations

using namespace std; // So we don't have to write "std::" before every library feature (like cout, cin, endl, etc)

class Exam { // Defining the Exam Class

private: // Access Modifier: private, makes member accessible only inside the class

char\* name; // Data Member: dynamically allocated memory for student name

char\* date; // Data Member: dynamically allocated memory for exam date

int score; // Data Member: storing exam score

public: // Access Modifier: public, allows functions to be accessible from outside

// Setters

void setScore(int x ) { // Method to set exam score

score = x ; // Assign new score

}

void setDate(const char\* x ) { // Method to set exam date

strcpy(date, x ); // Copy new date into allocated memory

}

void setName(const char\* x) { // Method to set student's name

if (name != nullptr) { // If memory already allocated, free it

delete[] name; // Free old memory before setting new name

}

name = new char[strlen( x ) + 1]; // Allocate memory for new name

strcpy(name, x ); // Copy new name into allocated memory

}

// Constructors

Exam(const char\* x, const char\* y, int z) { // Parameterized Constructor

name = new char[strlen( x ) + 1]; // Allocating memory for the student name

strcpy(name, x ); // Copying student name into the allocated memory

date = new char[strlen( y ) + 1]; // Allocating memory for the exam date

strcpy(date, y ); // Copying exam date into the allocated memory

score = z; // Assigning the exam score

cout << "Exam Initialized!" << endl<<endl; // Message when exam is initialized

}

// Display Method

void displayDetails() const { // Method to display student's exam details

cout << "Student: " << name << endl << "Exam Date: " << date << endl <<"Score: " << score << endl; // Print details of the exam

}

// Destructor

~Exam() { // Destructor to free dynamically allocated memory

string cleanupMessage = (name) ? name : "Unknown OR Nullptr"; // Prepare the cleanup message

cout << "Cleaning up memory for: " << cleanupMessage << endl; // Display cleanup message

delete[] name; // Free memory for name

delete[] date; // Free memory for exam date

}

};

int main() {

// Creating Exam Object Using Parameterized Constructor

cout << "----------------------------------" << endl;

cout << "Creating Exam Object student1" << endl;

Exam student1("Sharjeel Memon", "2025-05-30", 85); // Creates Exam object with initial values

student1.displayDetails(); // Display the details of student1

cout << endl;

// Creating a Shallow Copy of student1

cout << "----------------------------------" << endl;

cout << "Creating a Shallow Copy of student1 into student2" << endl;

Exam student2 = student1; // Creates student2 as a shallow copy of student1

student2.displayDetails(); // Display the details of student2

cout << endl;

// Case Number 1: Shallow Copy Issue - Dangling Pointer

cout << "----------------------------------" << endl;

cout << "Case Number 1: Dangling Pointer Issue when name is changed" << endl;

student2.setName("Muhammad Haneef"); // Changing the name of student2

student2.displayDetails(); // Display student2's details after name change

cout << "Student1 Details After student2 Name Change" << endl << endl;

student1.displayDetails(); // student1 shows garbage value for the name (dangling pointer)

cout << endl;

// Case Number 2: Shared Memory Issue - Exam Date is shared

cout << "----------------------------------" << endl;

cout << "Case Number 2: Both objects share the same memory for exam date" << endl;

student2.setDate("2025-06-15"); // Changing the exam date of student2

student2.displayDetails(); // Display student2's details after changing the date

cout << endl;

cout << "Student1 Details After student2 Date Change" << endl;

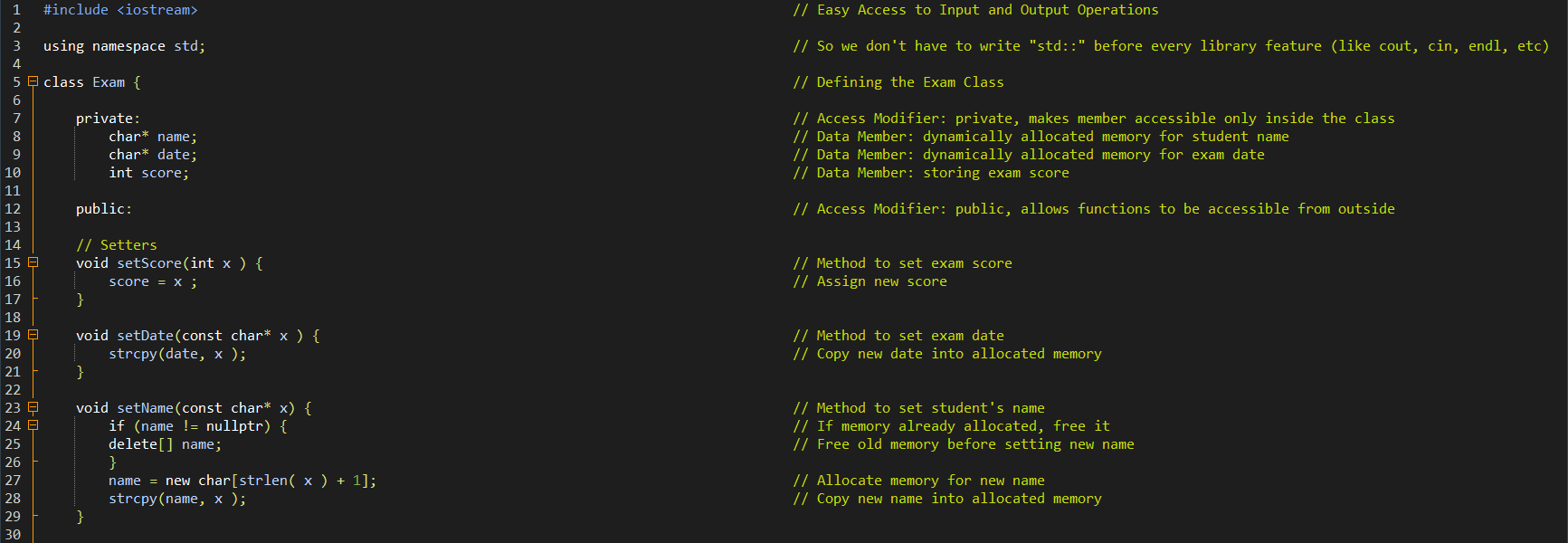
student1.displayDetails(); // student1's exam date also changes as they share the same memory

cout << endl << endl;

return 0; // End of Program

}

# Q2. Screenshots A screenshot of a computer AI-generated content may be incorrect.

 A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect. A black screen with yellow text

AI-generated content may be incorrect.

# Q3. Box Class with Dynamic Memory Allocation and Rule of Three

#include <iostream> // Easy Access to Input and Output Operations

using namespace std; // So we don't have to write "std::" before every library feature (like cout, cin, endl, etc)

class Box { // Defining the Box Class

private : // Access Modifier, private, makes member accessible only inside the class

int\* BoxSize; // Data Member: dynamically allocated memory for BoxSize

public : // Access Modifier, public, allows functions to be accessible from outside

// Constructors

Box() : BoxSize(new int(0)) { // Default Constructor - Initializes BoxSize with 0 if no parameter is passed

cout << "Box Created with default size!" << endl; // Message when the box is created

}

Box(int size) : BoxSize(new int(size)) { // Parameterized Constructor - Initializes BoxSize with the given size

cout << "Box Created with size " << \*BoxSize << endl; // Message when the box is created with a specified size

}

Box(const Box &other, bool deepCopy = true) { // Copy Constructor - Used to Create a copy of another Object

if (deepCopy) {

BoxSize = new int(\*(other.BoxSize)); // Deep copy: allocating new memory and copying the value of BoxSize

cout << "Box Created by copying with size " << \*BoxSize << endl; // Message when a box is created by copying another box

} else {

BoxSize = other.BoxSize; // Shallow copy: both objects point to the same memory

cout << "Box Created by shallow copying with size " << \*BoxSize << endl; // Message when a box is created by shallow copying another box

}

}

// Methods

void setBoxSize(int size) { // Method to set the size of the Box

\*BoxSize = size; // Assigning the given size to BoxSize

}

int getBoxSize() const { // Getter Method - Allows main to Access the Private Member

return \*BoxSize; // Returning the size of the Box

}

// Copy Assignment Operator - Used to assign values from one object to another

Box& operator=(const Box& other) { // Assignment operator to handle assignment between two Box objects

bool deepCopy = true; // Set to true for deep copy, false for shallow copy

if (this != &other) { // Check for self-assignment

delete BoxSize; // Free the existing memory

if (deepCopy) {

BoxSize = new int(\*(other.BoxSize)); // Perform deep copy

cout << "Box Assigned with size " << \*BoxSize << endl; // Message after deep copy assignment

} else {

BoxSize = other.BoxSize; // Perform shallow copy

cout << "Box Assigned by shallow copy with size " << \*BoxSize << endl; // Message after shallow copy assignment

}

}

return \*this; // Returning the current object

}

// Destructor

~Box() { // Destructor to free dynamically allocated memory

delete BoxSize; // Free memory for BoxSize

cout << "Box Destroyed" << endl; // Message when the box is destroyed

}

};

int main() {

// Deep Copy Example (Copy Constructor with deepCopy = true)

cout << "---- Deep Copy Example ----" << endl;

Box box1(10); // Will initialize with BoxSize 10

Box box2(box1, true); // Deep copy using the flag (true for deep copy)

box2.setBoxSize(20); // Modify box2

cout << "Box1 Size: " << box1.getBoxSize() << endl; // Should remain 10

cout << "Box2 Size: " << box2.getBoxSize() << endl << endl; // Should be 20

// Shallow Copy Example (Copy Constructor with deepCopy = false)

cout << "---- Shallow Copy Example ----" << endl;

Box box3(30);

Box box4(box3, false); // Shallow copy using the flag (false for shallow copy)

box4.setBoxSize(40); // Modify box4

cout << "Box3 Size: " << box3.getBoxSize() << endl; // Should change to 40 (shared memory)

cout << "Box4 Size: " << box4.getBoxSize() << endl << endl; // Should also be 40 (shared memory)

// Copy Assignment Example (Deep Copy)

cout << "---- Copy Assignment Example (Deep Copy) ----" << endl;

Box box5(50);

box5 = box1; // Deep copy using the assignment operator

cout << "Box5 Size: " << box5.getBoxSize() << endl << endl; // Should be 10

// Copy Assignment Example (Shallow Copy)

cout << "---- Copy Assignment Example (Shallow Copy) ----" << endl;

Box box6(60);

box6 = box3; // Shallow copy using the assignment operator (box3 and box6 will share memory)

box6.setBoxSize(70); // Modify box6

cout << "Box3 Size: " << box3.getBoxSize() << endl; // Should be 70 (shared memory with box6)

cout << "Box6 Size: " << box6.getBoxSize() << endl << endl; // Should be 70

return 0; // End of Program

}

# Q3 . Screenshots

A screenshot of a computer screen

AI-generated content may be incorrect. A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect. A black background with yellow dots

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.